MemoryAnalyzer

Contents

About

The **Eclipse Memory Analyzer (http://eclipse.org/mat)** tool (MAT) is a fast and feature-rich heap dump analyzer that helps you find **memory leaks** and analyze high **memory consumption** issues.

With Memory Analyzer one can easily

- find the biggest objects, as MAT provides reasonable accumulated size (retained size)
- explore the object graph, both inbound and outbound references
- compute paths from the garbage collector roots to interesting objects
- find memory waste, like redundant String objects, empty collection objects, etc...

Getting Started

Installation

See the **download page (http://eclipse.org/mat/downloads.php)** for installation instructions.

Basic Tutorials

Both the **Basic Tutorial (http://help.eclipse.org/neon/topic/org.eclipse.mat.ui.help/gettingstarted/basictutorial.html?cp=49_1_0)** chapter in the MAT documentation and the **Eclipse Memory Analyzer Tutorial (http://www.vogella.com/articles/EclipseMemoryAnalyser/article.html)** by Lars Vogel are a good first reading, if you are just starting with MAT.

Further Reading

Check **MemoryAnalyzer/Learning Material (/MemoryAnalyzer/Learning_Material)**. You will find there a collection of presentations and web articles on Memory Analyzer, which are also a good resource for learning.

Getting a Heap Dump

HPROF dumps from Sun Virtual Machines

The Memory Analyzer can work with *HPROF binary formatted heap dumps*. Those heap dumps are written by Sun HotSpot and any VM derived from HotSpot. Depending on your scenario, your OS platform and your JDK version, you have different options to acquire a heap dump.

Non-interactive

If you run your application with the VM flag **-XX:+HeapDumpOnOutOfMemoryError** a heap dump is written on the first Out Of Memory Error. There is no overhead involved unless a OOM actually occurs. This flag is a must for production systems as it is often the only way to further analyze the problem.

As per **this article (http://stackoverflow.com/questions/542979/using-heapdumponoutofmemoryerror-parameter-for-heap-dump-for-jboss)**, the heap dump will be generated in the "current directory" of the JVM by default. It can be explicitly redirected with **-XX:HeapDumpPath=** for example -*XX:HeapDumpPath=/disk2/dumps* . Note that the dump file can be huge, up to Gigabytes, so ensure that the target file system has enough space.

Interactive

As a developer, you want to trigger a heap dump on demand. On **Windows, use JDK 6 and JConsole**. On **Linux and Mac OS X**, you can also use **jmap** that comes with JDK 5.

Via MAT:

- tutorial **here (http://community.bonitasoft.com/blog/effective-way-fight-duplicated-libs-and-version-conflicting-classes-using-memory-analyzer-tool)**

Via Java VM parameters:

- -XX:+HeapDumpOnOutOfMemoryError writes heap dump on OutOfMemoryError (recommended)
- -XX:+HeapDumpOnCtrlBreak writes heap dump together with thread dump on CTRL+BREAK
- -agentlib:hprof=heap=dump,format=b combines the above two settings (old way; not recommended as the VM frequently dies after CTRL+BREAK with strange errors)

Via Tools:

- Sun (Linux, Solaris; not on Windows) **JMap Java 5 (http://java.sun.com/j2se/1.5.0/docs/tooldocs/share/jmap.html)**: **jmap -heap:format=b <pid>**
- Sun (Linux, Solaris; Windows see link) **JMap Java 6 (http://java.sun.com/javase/6/docs/technotes/tools/share/jmap.html)**: **jmap.exe -dump:format=b,file=HeapDump.hprof <pid>**
- Sun (Linus, Solaris) JMap with Core Dump File: **jmap -dump:format=b,file=HeapDump.hprof /path/to/bin/java core_dump_file**
- Sun JConsole: Launch jconsole.exe and invoke operation dumpHeap() on HotSpotDiagnostic MBean
- SAP JVMMon: Launch jvmmon.exe and call menu for dumping the heap

Heap dump will be written to the working directory.

Vendor / Release	VM Parameter			VM Tools	
	On OoM	On Ctrl+Break	Agent	JMap	JConsole
Sun, HP					
1.4.2_12	Yes	Yes	Yes		
1.5.0_07	Yes	Yes (Since 1.5.0_15)	Yes	Yes (Only Solaris and Linux)	
1.6.0_00	Yes		Yes	Yes	Yes
SAP					
1.5.0_07	Yes	Yes	Yes	Yes (Only Solaris and Linux)	

System Dumps and Heap Dumps from IBM Virtual Machines

Memory Analyzer may read memory-related information from IBM system dumps and from Portable Heap Dump (PHD) files with the **IBM DTFJ feature (http://www.ibm.com/developerworks/java/jdk/tools/dtfj.html)** installed. Once installed, then **File** > **Open Heap Dump** should give the following options for the file types:

- All known formats
- HPROF binary heap dumps
- IBM 1.4.2 SDFF
- IBM Javadumps
- IBM SDK for Java (J9) system dumps
- IBM SDK for Java Portable Heap Dumps

For a comparison of dump types, see **Debugging from dumps (http://www.ibm.com/developerworks/library/j-memoryanalyzer/#table1)**. System dumps are simply operating system core dumps; therefore, they are a superset of portable heap dumps. System dumps are far superior than PHDs, particularly for more accurate GC roots, thread-based analysis, and unlike PHDs, system dumps contain memory contents like HPROFs. Older versions of IBM Java (e.g. < 5.0SR12, < 6.0SR9) require running jextract on the operating system core dump which produced a zip file that contained the core dump, XML or SDFF file, and shared libraries. The IBM DTFJ feature still supports reading these jextracted zips; however, newer versions of IBM Java do not require jextract for use in MAT since DTFJ is able to directly read each supported operating system's core dump format. Simply ensure that the operating system core dump file ends with the **.dmp** suffix for visibility in the MAT Open Heap Dump selection. It is also common to zip core dumps because they are so large and compress very well. If a core dump is compressed with **.zip**, the IBM DTFJ feature in MAT is able to decompress the ZIP file and read the core from inside (just like a jextracted zip). The only significant downsides to system dumps over PHDs is that they are much larger, they usually take longer to produce, they may be useless if they are manually taken in the middle of an exclusive event that manipulates the underlying Java heap such as a garbage collection, and they sometimes require operating system configuration (**Linux (http://www.ibm.com/support/knowledgecenter/SSYKE2_7.1.0/com.ibm.java.lnx.71.doc/diag/problem_determination/linux_setup.html)**, **AIX (http://www.ibm.com/support/knowledgecenter/SSYKE2_7.1.0/com.ibm.java.aix.71.doc/diag/problem_determination/aix_setup_full_core.html)**) to ensure non-truncation.

In recent versions of IBM Java (> 6.0.1), by default, when an OutOfMemoryError is thrown, IBM Java **produces (http://www.ibm.com/support/knowledgecenter/SSYKE2_8.0.0/com.ibm.java.lnx.80.doc/diag/tools/dumpagents_defaults.html)** a system dump, PHD, javacore, and Snap file on the first occurrence for that process (although often the core dump is suppressed by the default 0 core ulimit on operating systems such as Linux). For the next three occurrences, it produces only a PHD, javacore, and Snap. If you only plan to use system dumps, and you've configured your operating system correctly as per the links above (particularly core and file ulimits), then you may disable PHD generation with -Xdump:heap:none. For versions of IBM Java older than 6.0.1, you may switch from PHDs to system dumps using -Xdump:system:events=systhrow,filter=java/lang/OutOfMemoryError,request=exclusive+prepwalk -Xdump:heap:none

In addition to an OutOfMemoryError, system dumps may be produced using operating system tools (e.g. gcore in gdb for Linux, gencore for AIX, Task Manager for Windows, SVCDUMP for z/OS, etc.), using the **IBM Java APIs (http://www.ibm.com/support/knowledgecenter/SSYKE2_8.0.0/com.ibm.java.lnx.80.doc/diag/tools/diagnostics_summary.html)**, using the various options of **-Xdump (http://www.ibm.com/support/knowledgecenter/SSYKE2_8.0.0/com.ibm.java.lnx.80.doc/diag/tools/dump_agents.html)**, using **Java Surgery (https://www.ibm.com/developerworks/community/groups/service/html/communityview?communityUuid=7d3dc078-131f-404c-8b4d-68b3b9ddd07a)**, and more.

Versions of IBM Java older than IBM JDK 1.4.2 SR12, 5.0 SR8a and 6.0 SR2 are known to produce inaccurate GC root information.

What if the Heap Dump is NOT Written on OutOfMemoryError?

Heap dumps are not written on OutOfMemoryError for the following reasons:

- Application creates and throws OutOfMemoryError on its own
- Another resource like threads per process is exhausted
- C heap is exhausted

As for the C heap, the best way to see that you won't get a heap dump is if it happens in C code (eArray.cpp in the example below):

```
# An unexpected error has been detected by SAP Java Virtual Machine:
# java.lang.OutOfMemoryError: requested 2048000 bytes for eArray.cpp:80: GrET*. Out of swap space or heap resource limit exceeded (check with
# Internal Error (\\...\hotspot\src\share\vm\memory\allocation.inline.hpp, 26), pid=6000, tid=468
```

C heap problems may arise for different reasons, e.g. out of swap space situations, process limits exhaustion or just address space limitations, e.g. heavy fragmentation or just the depletion of it on machines with limited address space like 32 bit machines. The hs_err-file will help you with more information on this type of error. Java heap dumps wouldn't be of any help, anyways.

Also please note that a heap dump is written only on the first OutOfMemoryError. If the application chooses to catch it and continues to run, the next OutOfMemoryError will never cause a heap dump to be written!

Extending Memory Analyzer

Memory Analyzer is extensible, so new queries and dump formats can be added. Please see **MemoryAnalyzer/Extending_Memory_Analyzer (/MemoryAnalyzer/Extending_Memory_Analyzer)** for details.